HoGent

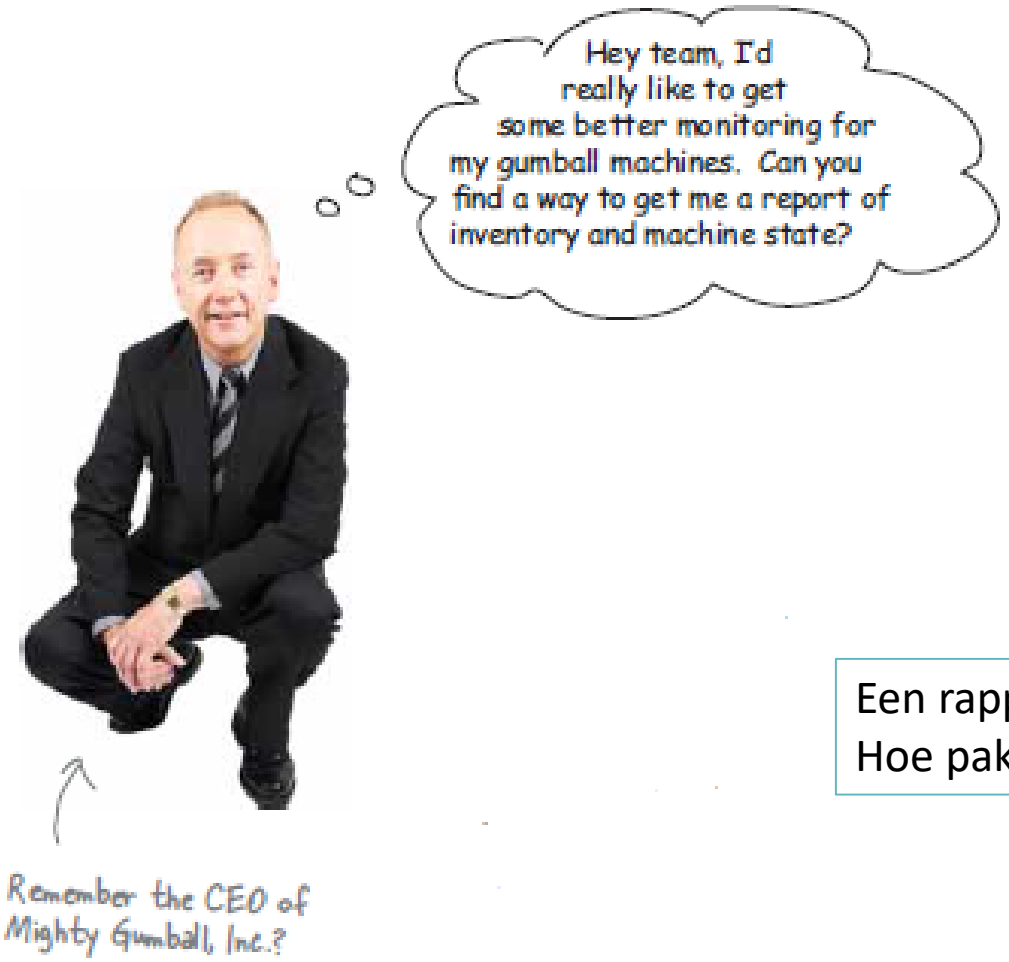
BEDRIJF
EN
ORGANISATIE

Het proxy pattern

De objecttoegang controleren

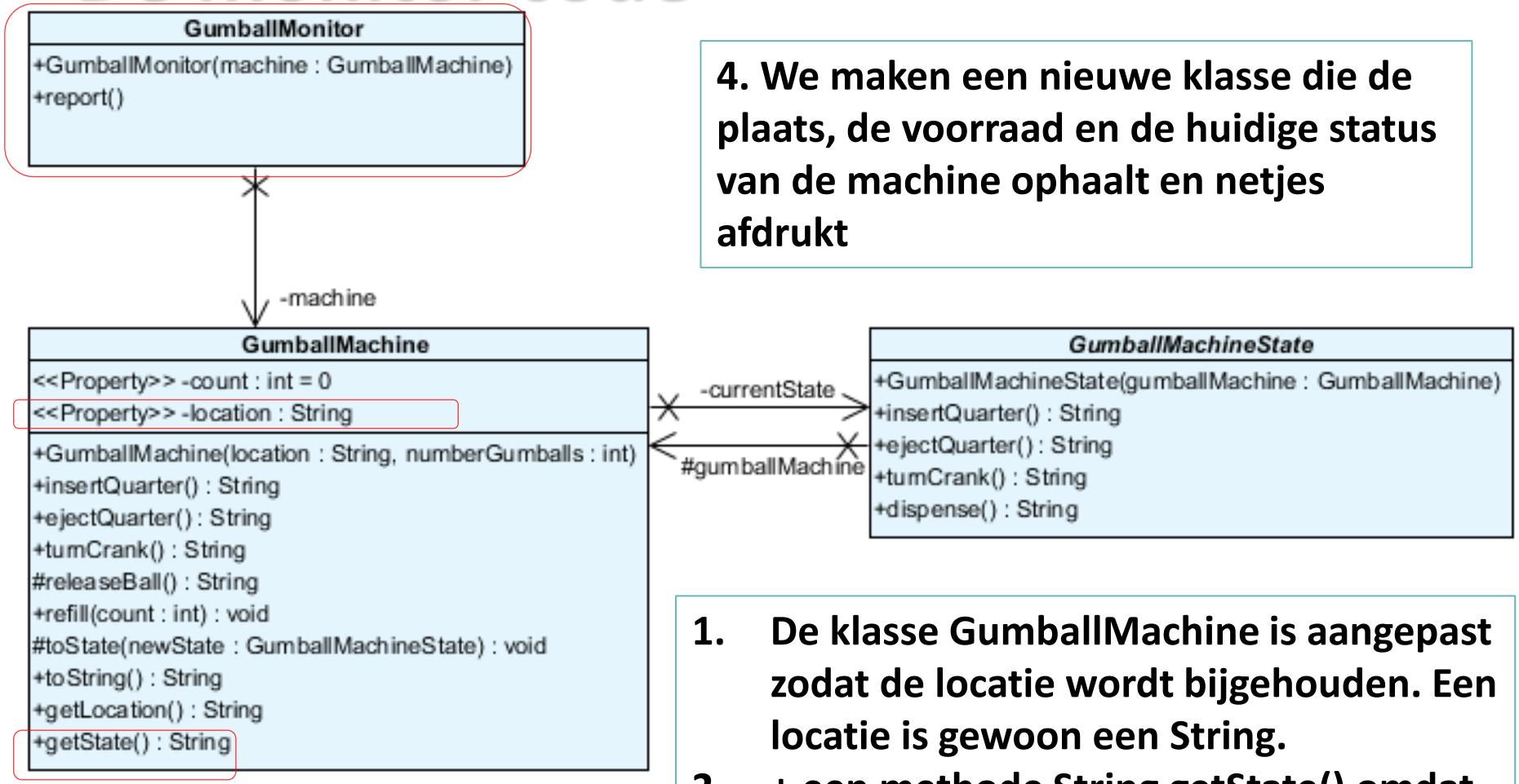
HoGent

De Gumball monitor



Een rapportje. Simpel.
Hoe pakken we het aan?

De monitor code



4. We maken een nieuwe klasse die de plaats, de voorraad en de huidige status van de machine ophaalt en netjes afdrukt

1. **De klasse GumballMachine is aangepast zodat de locatie wordt bijgehouden. Een locatie is gewoon een String.**
2. **+ een methode String getState() omdat de monitor info over de “toestand” van de automaat wil.**
3. **Methode getCount() levert de voorraad**

De monitor code

```
public class GumballMonitor {

    private GumballMachine machine;

    public GumballMonitor(GumballMachine machine) {
        this.machine = machine;
    }

    public void report() {
        System.out.println("Gumball machine: " + machine.getLocation());
        System.out.println("Current inventory: " + machine.getCount());
        System.out.println("Current state: " + machine.getState());
    }
}
```

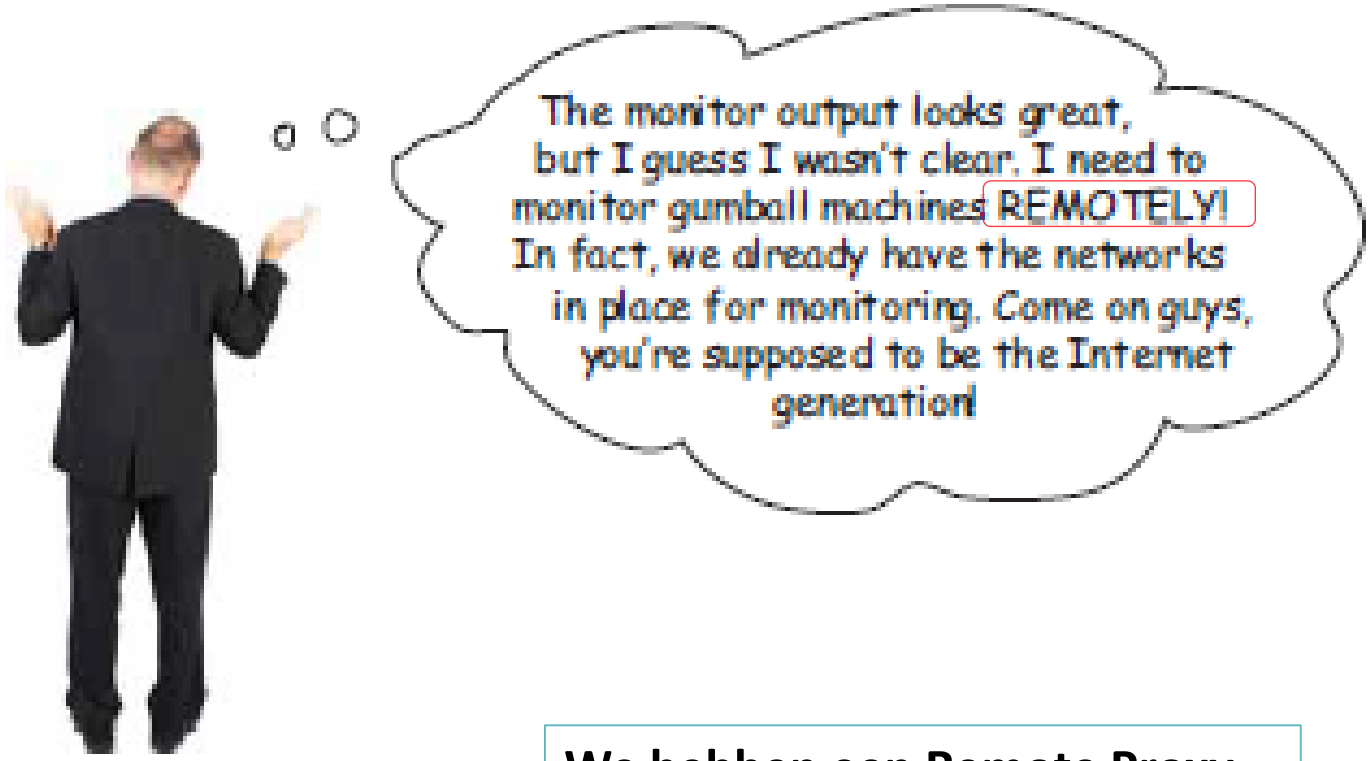
Het rapportje : met de plaats, voorraad en machine status

De monitor uitproberen

```
public static void main(String[] args) {
    int count = 0;
    if (args.length < 2) {
        System.out.println("Gumball machine <name> <inventory>");
        System.exit(1);
    }

    count = Integer.parseInt(args[1]);
    GumballMachine machine = new GumballMachine(args[0], count);
    GumballMonitor monitor = new GumballMonitor(machine);
    monitor.report();
}
```

```
Gumball machine: HoGent
Current inventory: 112
Current state: NoQuarterState
```

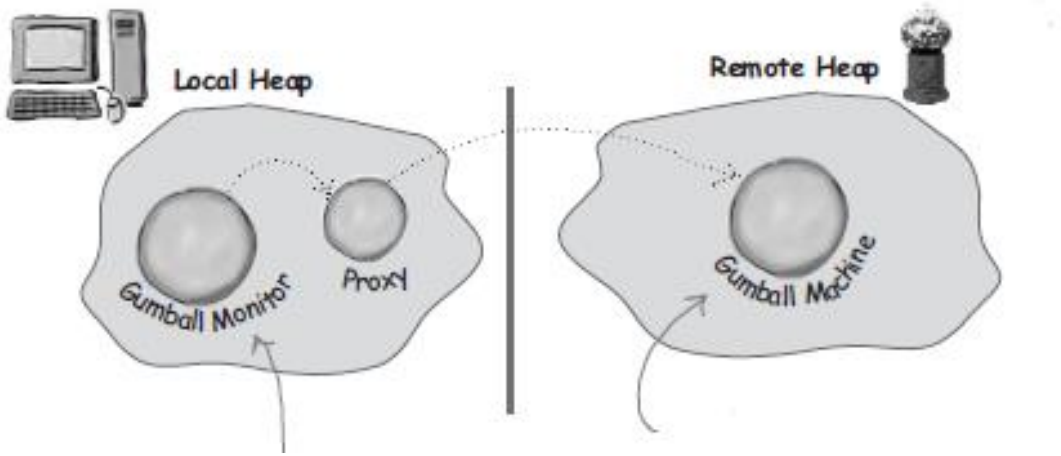
De monitor uitproberen



We hebben een Remote Proxy nodig, en we kunnen verder

De rol van de 'remote proxy'

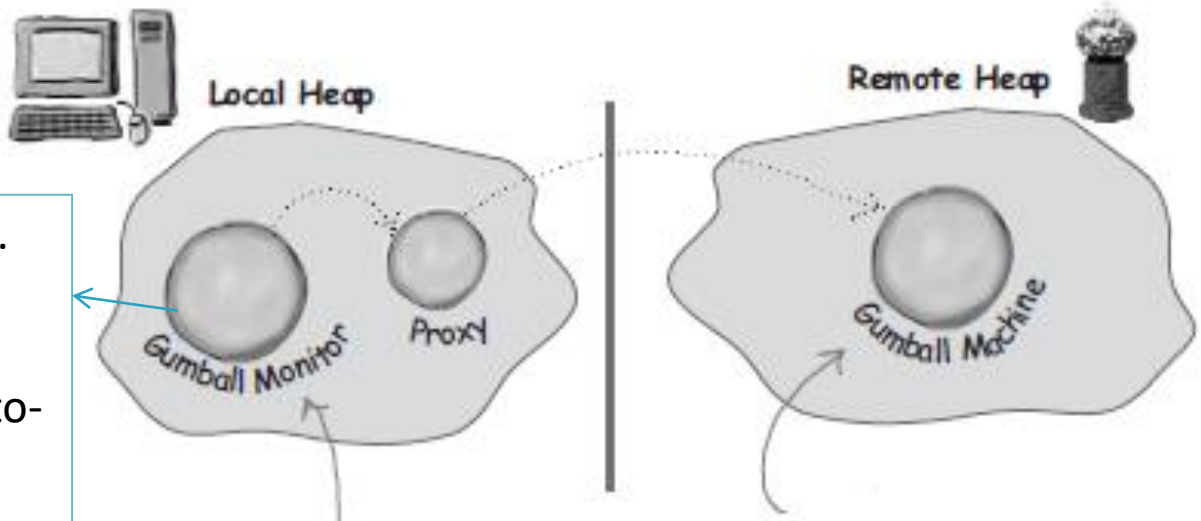
- **Remote proxy** fungeert als een lokale vertegenwoordiger van een remote object.
  - Draait in een ander JVM (andere adresruimte)
  - Het bevat een aantal lokale methoden die aangeroepen kunnen worden en die het vervolgens doorstuurt naar het remote object



De rol van de 'remote proxy'

Desktop van de CEO

Remote GumballMachine met een JVM



Is een client object. Hij denkt dat hij tegen de echte kauwgomballenauto-maat(KA) babbelt, maar hij praat eigenlijk tegen de proxy, die dan weer over het netwerk praat met de echte KA

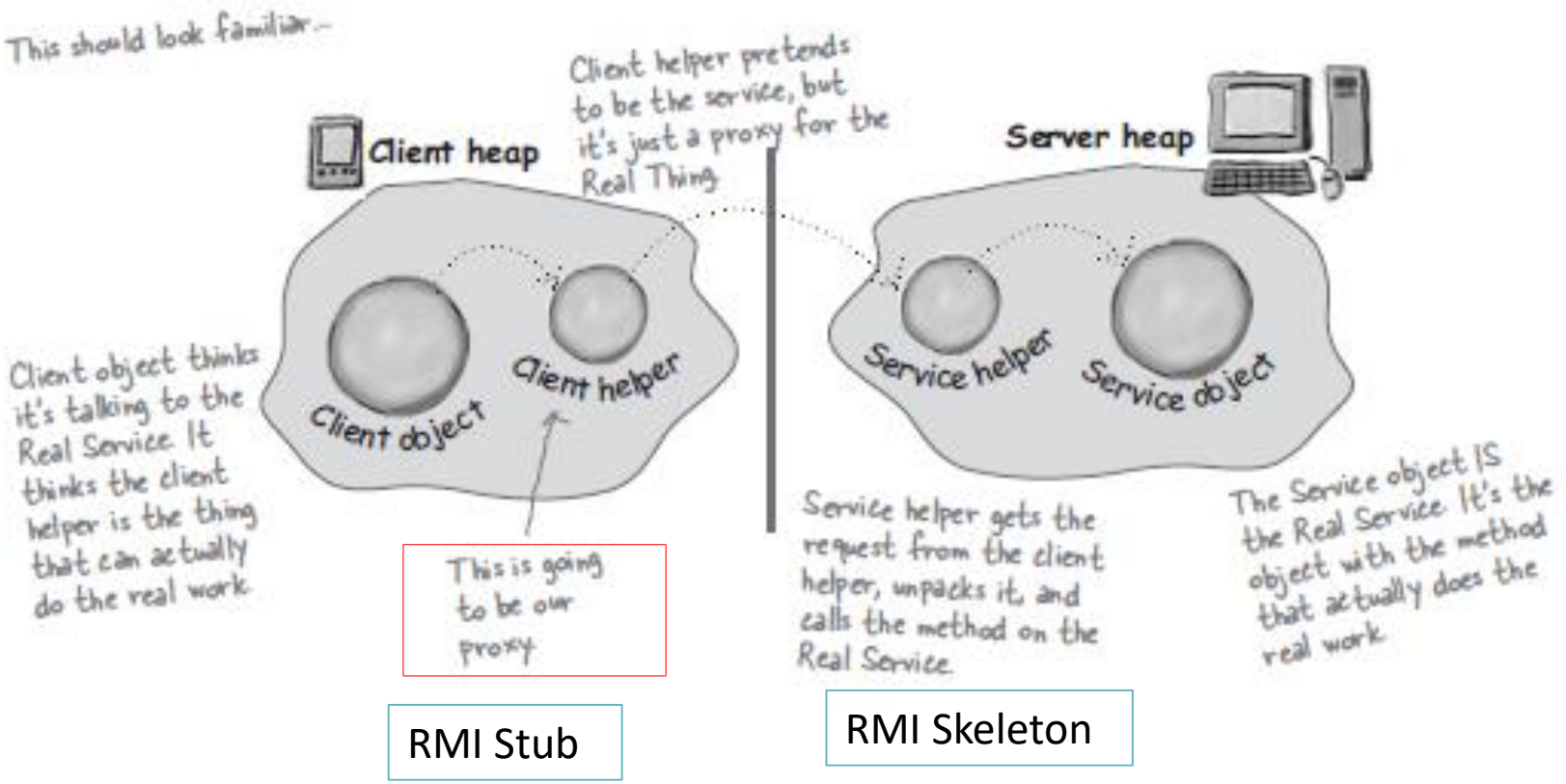
Zelfde als in de oude code, alleen praat hij nu tegen de proxy

Het remote object IS DE REAL THING. Het is het objectje met de methode die feitelijk het echte werk doet

Het lijkt alsof client remote methode-aanroepen pleegt. In werkelijkheid worden methoden aangeroepen van proxy object in lokale heap, die dan alle low-level details van de netwerkcommunicatie afhandelt

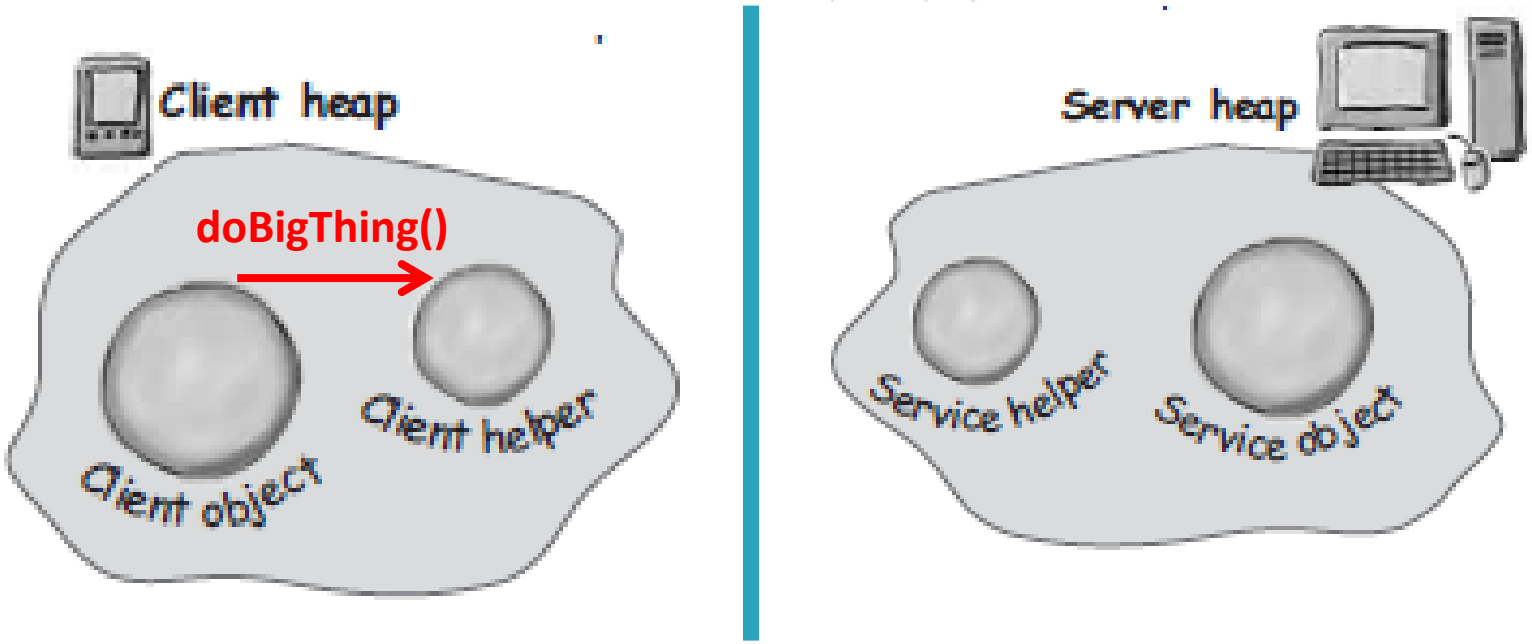
R(emote) M(ethod) I(nvocation)

- Laat toe om objecten te vinden in een remote JVM en hun methoden aan te roepen



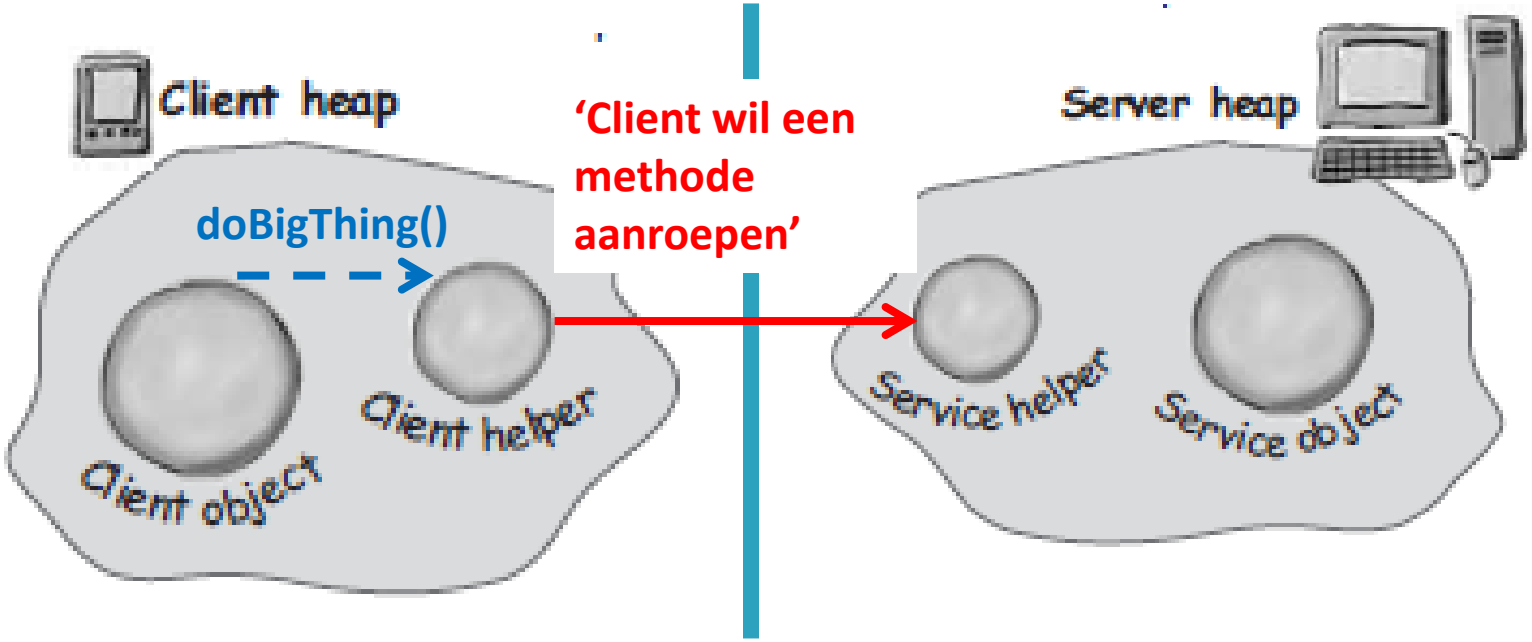
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 1. Client object roept de methode doBigThing() aan van het clienthulp object



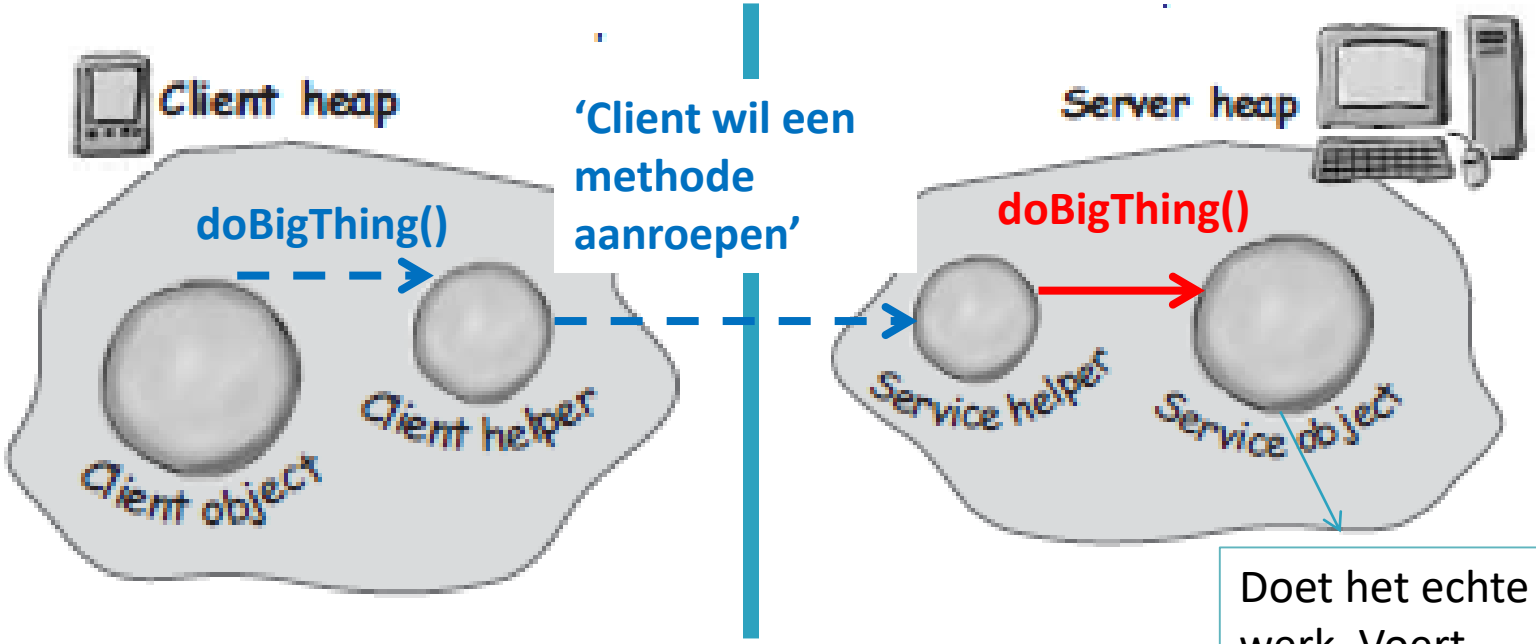
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 2. Clienthulp verpakt de info over de aanroep (methodenaam, argumenten, …) en stuurt deze over het netwerk naar service hulp



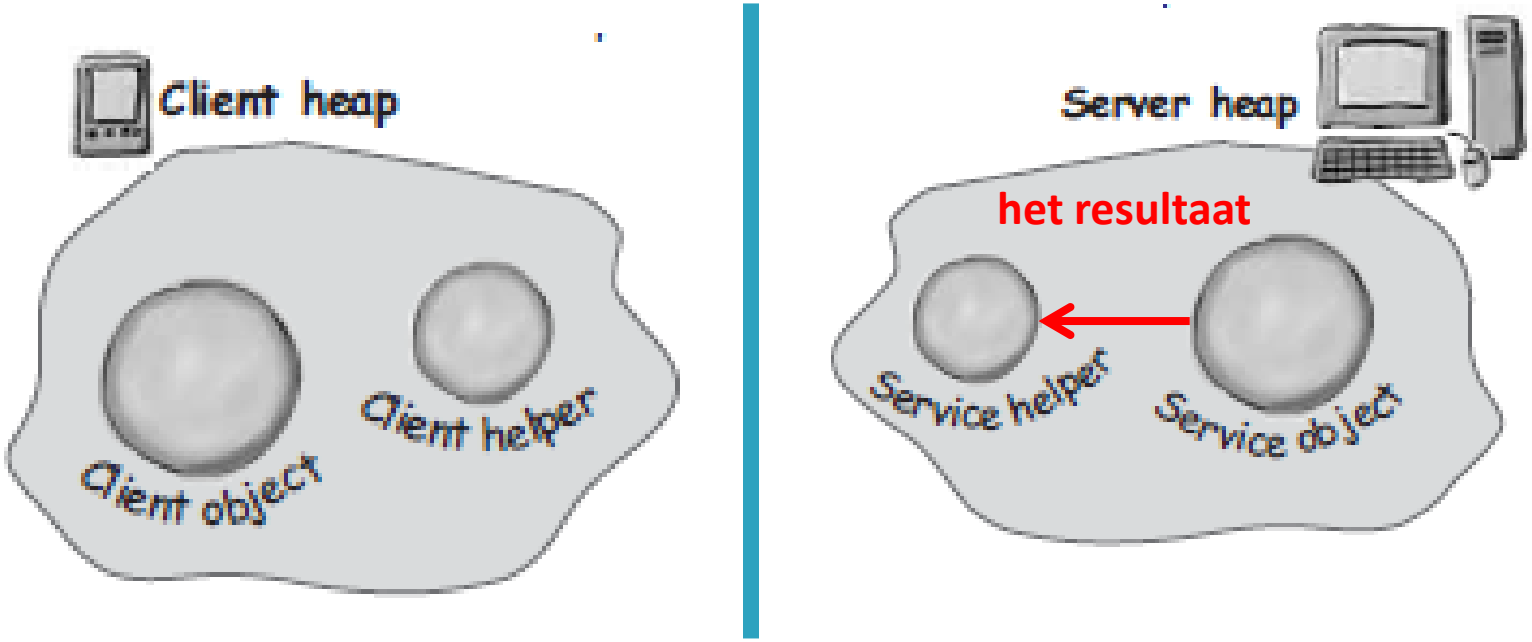
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 3. Servicehulp pakt de info uit, ondekt de methode die moet worden aangeroepen (en voor welk object) en roept de ECHTE methode voor het ECHTE service object aan



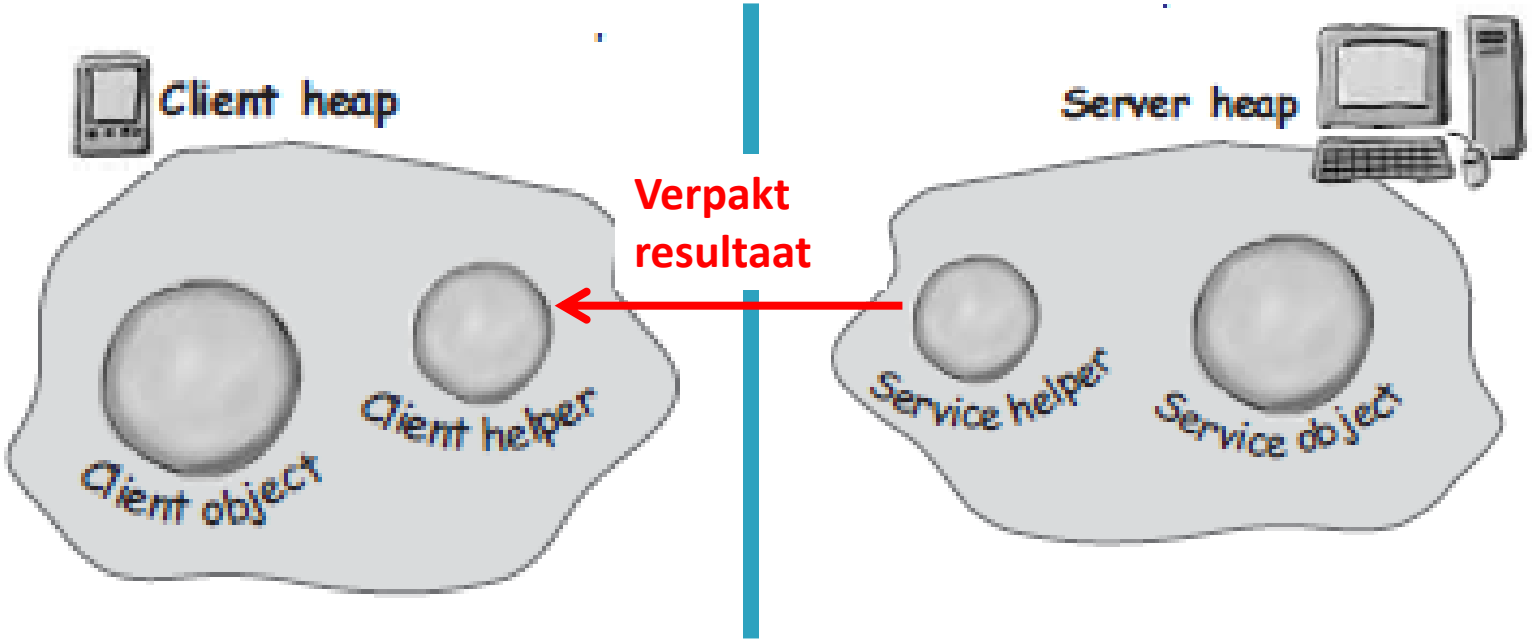
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 4. Het Service object voert de methode doBigThing() uit en retourneert het resultaat naar de Servicehulp



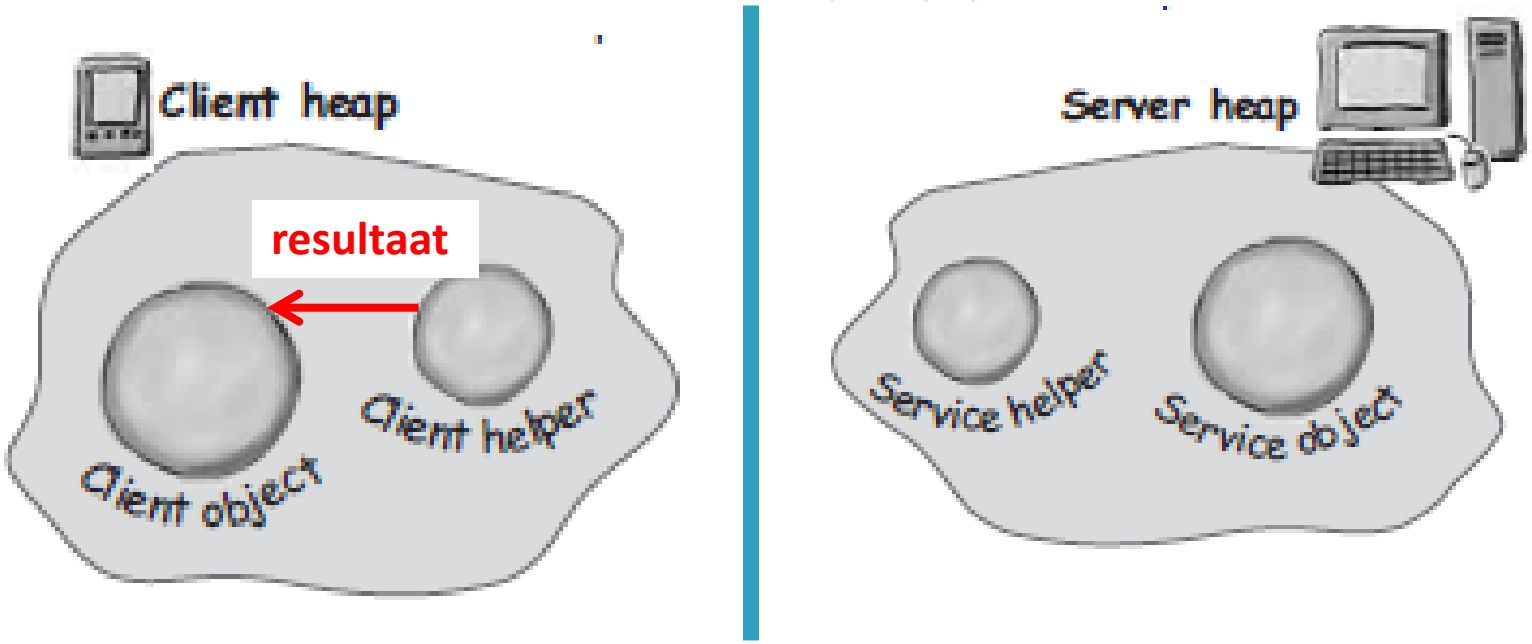
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 5. De Servicehulp verpakt het resultaat en verstuurt dit over het netwerk naar de clienthulp



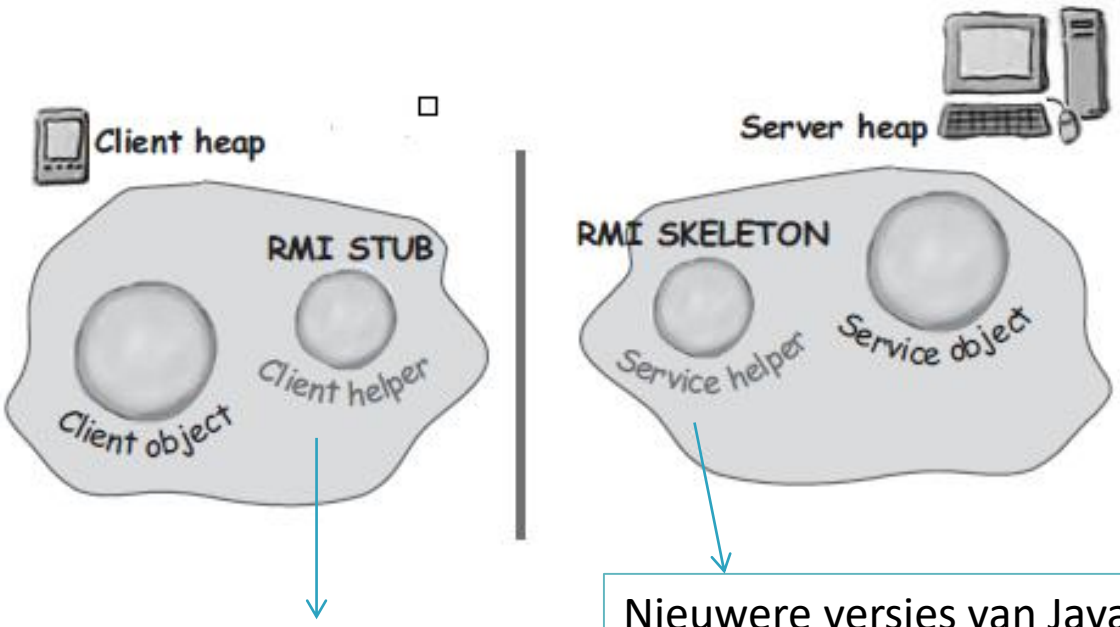
RMI uitstapje

- Wat gebeurt er bij een methode aanroep?
  - 6. De Clienthulp pakt het resultaat uit en levert het aan het client object. Dit alles was transparant voor het client object.



Java RMI, het totaalbeeld

- RMI-nomenclatuur : in RMI is de clienthulp een “stub” en de servicehulp een “skeleton”. RMI bouwt beide

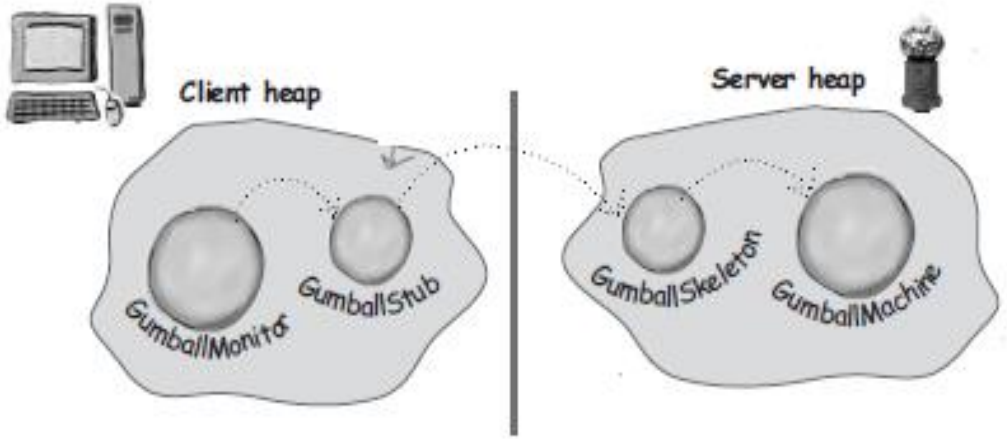


Deze gaat fungeren als onze proxy

Nieuwere versies van Java vereisen geen explicit skeleton-object, maar iets aan de serverkant handelt nog steeds het skeletongedrag af

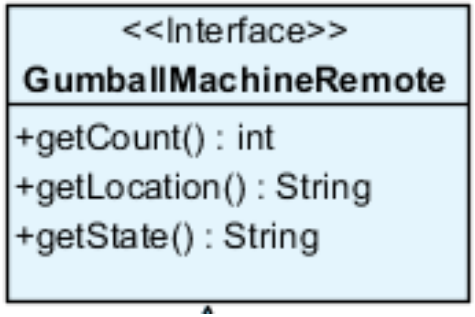
De Remote Service maken

- Stap 1 : Remote interface maken
- Stap 2 : Remote implementatie maken
- Stap 3 : stub en skeleton worden dynamisch aangemaakt door de JVM (sinds java1.5)
- Stap 4 : RMI registry starten (= name service voor remote objecten)
- Stap 5 : Het remote object bekend maken bij de name service
- Stap 6 : Run



GumballMachine klaarzetten als remote service

- Stap 1 : Remote interface maken
  - Definieert de remote methoden die de client kan aanroepen. Zowel de stub als actuele service implementeren deze
    - Breid Java.rmi.Remote uit
    - Declareer dat alle methoden een RemoteException afgeven
    - Zorg ervoor dat alle argumenten en retourwaarden primitief of serialiseerbaar zijn




```
import java.rmi.*;

public interface GumballMachineRemote extends Remote {

   int getCount() throws RemoteException;

   String getLocation() throws RemoteException;

   String getState() throws RemoteException;

}
```

GumballMachine klaarzetten als remote service

- Stap 1 : Remote interface maken
  - Probleem 1 : Alle argumenten en retourwaarden moeten serialiseerbaar zijn.
    - Stel dat we een State object retourneren, dan maak je dit als volgt serialiseerbaar :

```
import java.io.Serializable;

public abstract class GumballMachineState implements Serializable {
    transient protected GumballMachine gumballMachine;
    protected GumballMachineState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }
    protected String insertQuarter() {
        return "You can't insert a quarter";
    }
```

GumballMachine klaarzetten als remote service

- Stap 1 : Remote interface maken
  - Probleem 2 : Wat als je niet het volledige object wenst te serialiseren, maar slechts een onderdeel ervan.
    Stel dat we terug een State object retourneren : iedere concrete State klasse bevat een referentie naar GumballMachine. We willen niet dat de hele GumballMachine geserialiseerd en overgedragen wordt met het State object
    - transient : zegt tegen de JVM dat dit veld niet geserialiseerd moet worden

```
public abstract class GumballMachineState implements Serializable {
    transient protected GumballMachine gumballMachine;
    protected GumballMachineState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }
    protected String insertQuarter() {
        return "You can't insert a quarter";
```

GumballMachine klaarzetten als remote service

- Stap 2 : Remote implementatie maken
  - Deze klasse doet het echte werk. Dit is het object waarvan de client de methoden wil aanroepen.
    - Implementeer de remote interface
    - Breid UnicastRemoteObject uit (zo kan het van buitenaf worden aangesproken)
      - Indien niet mogelijk (Geen multiple Inheritance), exporteer het remoteObject:
        - stub = (RemoteInterface)
        - UnicastRemoteObject.exportObject(RemoteObject, 0);
    - Schrijf een constructor die een remoteException declareert

GumballMachine klaarzetten als remote service

- Stap 2 :

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class GumballMachine extends UnicastRemoteObject
                                implements GumballMachineRemote {
  private GumballMachineState currentState;
  private int count = 0;
  private final String location;
  public GumballMachine(String location, int numberGumballs)
                                        throws RemoteException {
    this.location = location;
    this.count = numberGumballs;
    toState( (numberGumballs > 0) ? new NoQuarterState(this)
                                  : new OutOfGumballsState(this));
  }

  public String insertQuarter() {
    return currentState.insertQuarter();
  }
```

GumballMachine klaarzetten als remote service

- Stap 4 en 5 : RMI service starten en het remote object bekend maken bij de name service
  - Stap 4 : Start eerst RMI registry service

```
Registry registry = LocateRegistry.createRegistry(1099);
```

    - Opm. Je kan dit ook starten via command prompt, commando rmiregistry (vanuit map die de map met de class files bevat)
  - Stap 5 : Stel je remote service beschikbaar aan de remote clients.
    - Hiervoor dien je deze te instantiëren en te plaatsen in de RMI registry (die wel moet draaien)
    - registry.rebind : Bij de registratie plaatst het RMI systeem een GumballMachine-**stub** in de registry met de naam gumballmachine. Dit is wat de client echt nodig heeft.

```
machine = new GumballMachine(location, count);
registry.rebind("gumballmachine", machine);
```

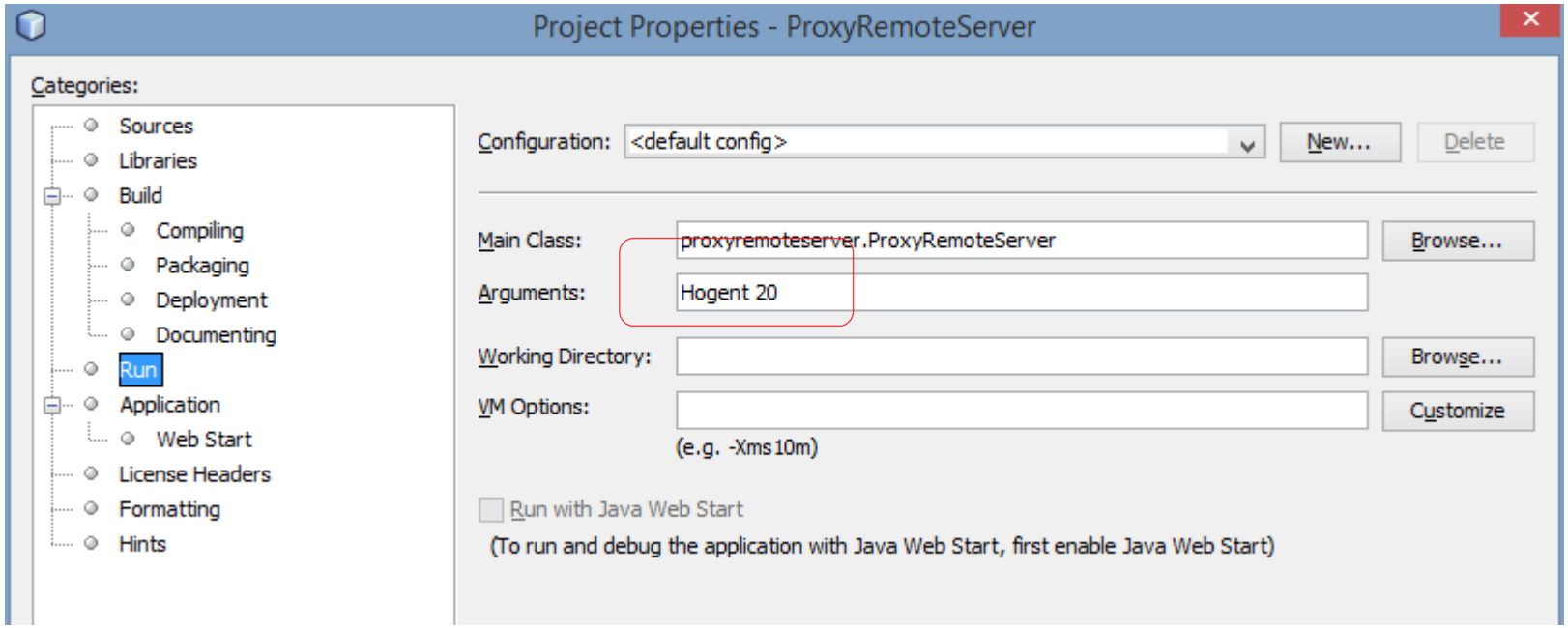
GumballMachine klaarzetten als remote service

- Stap 4 en 5
  - Meld de service aan bij de RMI registry
    - Voeg try…catch toe, zodat constructor exceptions kan afgeven

```
private void registerRemoteGumballMachine() {
   try {
      // create remoteobject nameservice on port 1099
      Registry registry = LocateRegistry.createRegistry(1099);
      machine = new GumballMachine(location, count);
      registry.rebind("gumballmachine", machine);
   } catch (RemoteException ex) {
      ex.printStackTrace();
   }
}
```

GumballMachine klaarzetten als remote service

- Stap 6 : Run
  - We dienen 2 parameters op te geven alvorens we het project starten
    - Run > Set project configuration > customize > Project options > run > Argument.



Een extraatje

- At random om de 1,5..12,5 sec wordt een kauwgom uit de automaat gehaald

```
private void run() {
   System.out.println("GumballMachine Operational.");
   System.out.println(machine);
   while (machine.getCount() > 0) {
      try {
         Thread.sleep(ThreadLocalRandom.current().nextInt(9000) + 1000);
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
      getAGumball();
      System.out.println(machine);
   }
}
```

Een extraatje

- At random om de 1,5..12,5 sec wordt een kauwgom uit de automaat gehaald

```
private void getAGumball() {
   System.out.println(machine.insertQuarter());
   try {
      Thread.sleep(ThreadLocalRandom.current().nextInt(2000) + 500);
   } catch (InterruptedException ex) {
      ex.printStackTrace();
   }
   System.out.println(machine.turnCrank());
}
```

```
GumballMachine Operational.

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 20 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 19 gumballs
Machine is waiting for quarter
```

De client van GumballMonitor

- Maak een nieuw project : ProxyRemoteClient
- Plaats een kopie van de GumballMachineRemote interface in dit project
- Plaats de GumballMonitor in dit project. De code moet worden aangepast om met de proxy te werken

De client van GumballMonitor

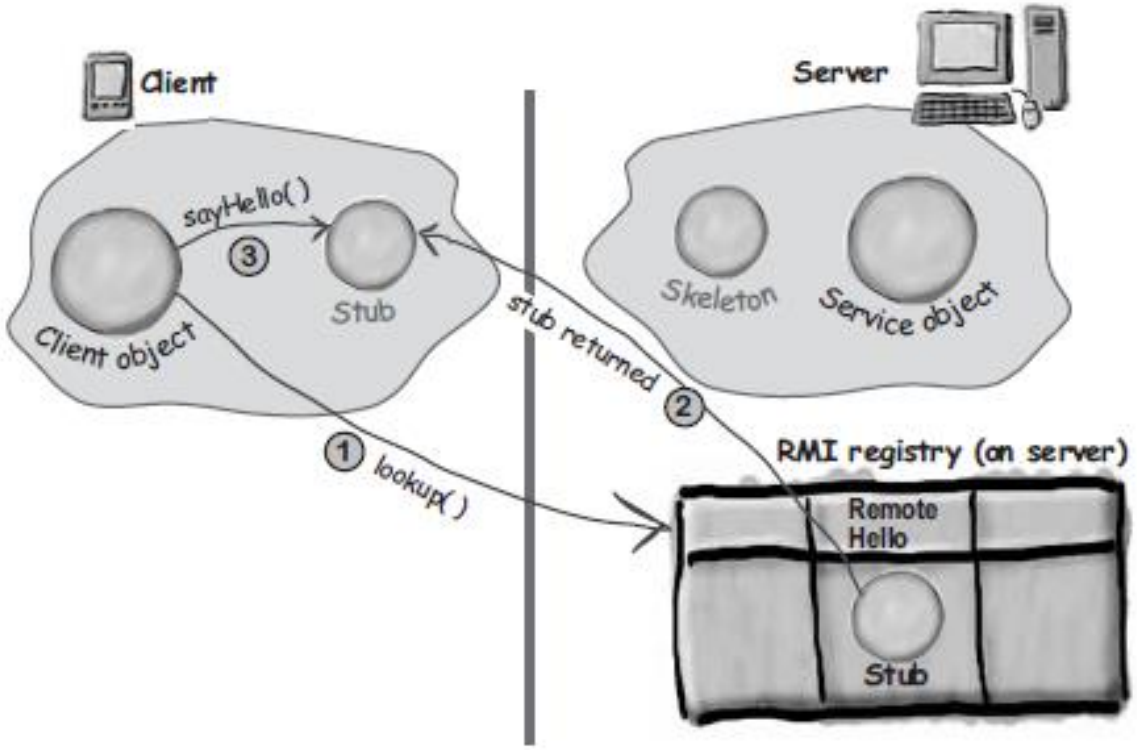
```
public class GumballMonitor {

    private GumballMachineRemote machine;

    public GumballMonitor(GumballMachineRemote machine) {
        this.machine = machine;
    }

    public void report() {
        try {
            //call remote services
            System.out.println("Gumball machine: " + machine.getLocation());
            System.out.println("Current inventory: " + machine.getCount());
            System.out.println("Current state: " + machine.getState());
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }
}
```

We werken nu met de remote interface ipv met de concrete klasse

We moeten de remote exceptions opvangen die zich kunnen voordoen als we proberen methoden aan te roepen die over het netwerk plaatsvinden

Hoe komt de client aan het stubobject

- Via de RMI Registry :
  - 1. Client zoekt RMI Registry
  - 2. RMI registry retourneert stubobject
  - 3. Client roept methode van stub aan, alsof stub de echte service is



Hoe komt de client aan het stubobject

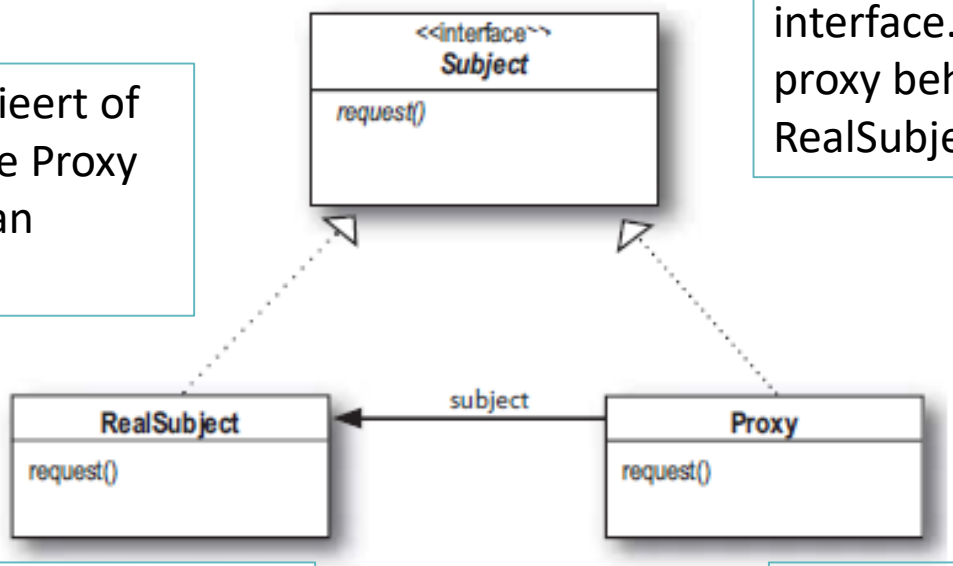
```
private void doTest() {
   try {
      // get remote registry object on port 1099 (rmi nameservice)
      Registry myRegistry = LocateRegistry.getRegistry("127.0.0.1", 1099);
      // dit is de default ==> LocateRegistry.getRegistry();

      // search for remote object GumballMachineRemote (via rmi nameservice)
      GumballMachineRemote machine =
            (GumballMachineRemote) myRegistry.lookup("gumballmachine");

      // Geef remote object door aan GumballMonitor
      GumballMonitor monitor = new GumballMonitor(machine);

      monitor.report();
   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

Het proxy pattern

Het proxy pattern zorgt voor een surrogaat of plaatsvervanger voor een ander object om de toegang hiertoe te controleren

Zowel de Proxy als RealSubject implementeren de Subject-interface. Zo kan iedere client de proxy behandelen als het RealSubject

Vaak instantieert of behandelt de Proxy de creatie van RealSubject

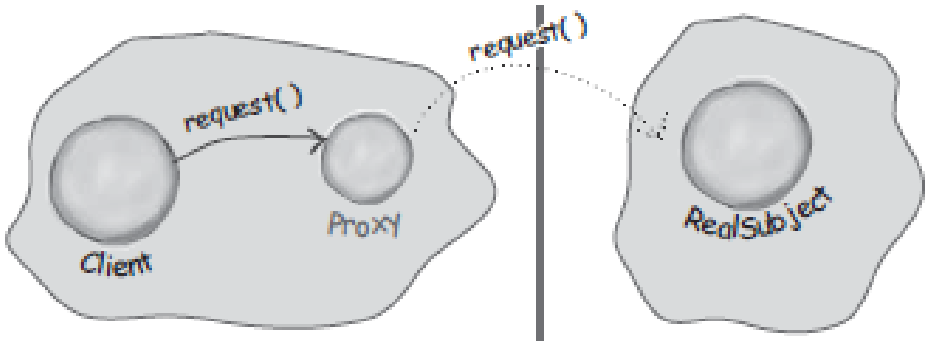


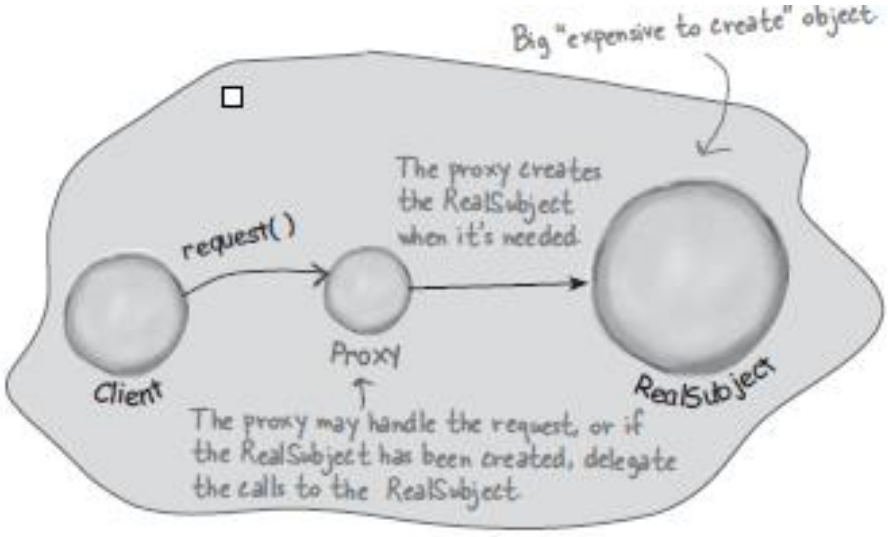
Het RealSubject is het object dat het meeste werkt verricht. De Proxy regelt de toegang tot dit object

De proxy bewaart een referentie naar het Subject, om zo, wanneer nodig, aanvragen naar het Subject door te sturen

De virtuele proxy

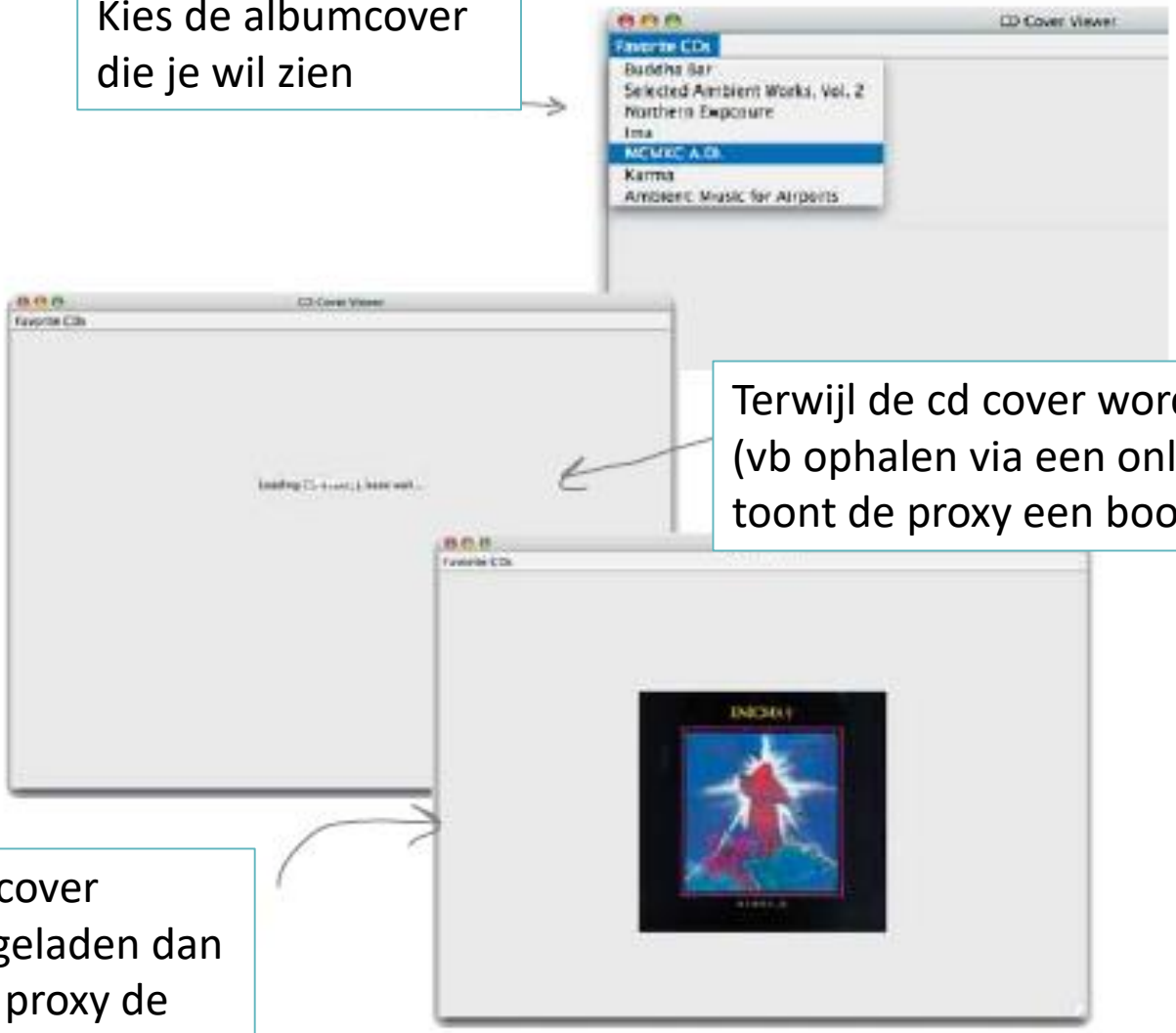
- Remote proxy
  - Lokale vertegenwoordiger van een object in een andere JVM
- Virtuele proxy
  - Vertegenwoordiger van object dat te duur kan zijn om zelf te maken. Proxy fungeert als surrogaat voor object voor en tijdens zijn creatie. Daarna delegeert proxy direct naar RealSubject





Een cd-cover tonen

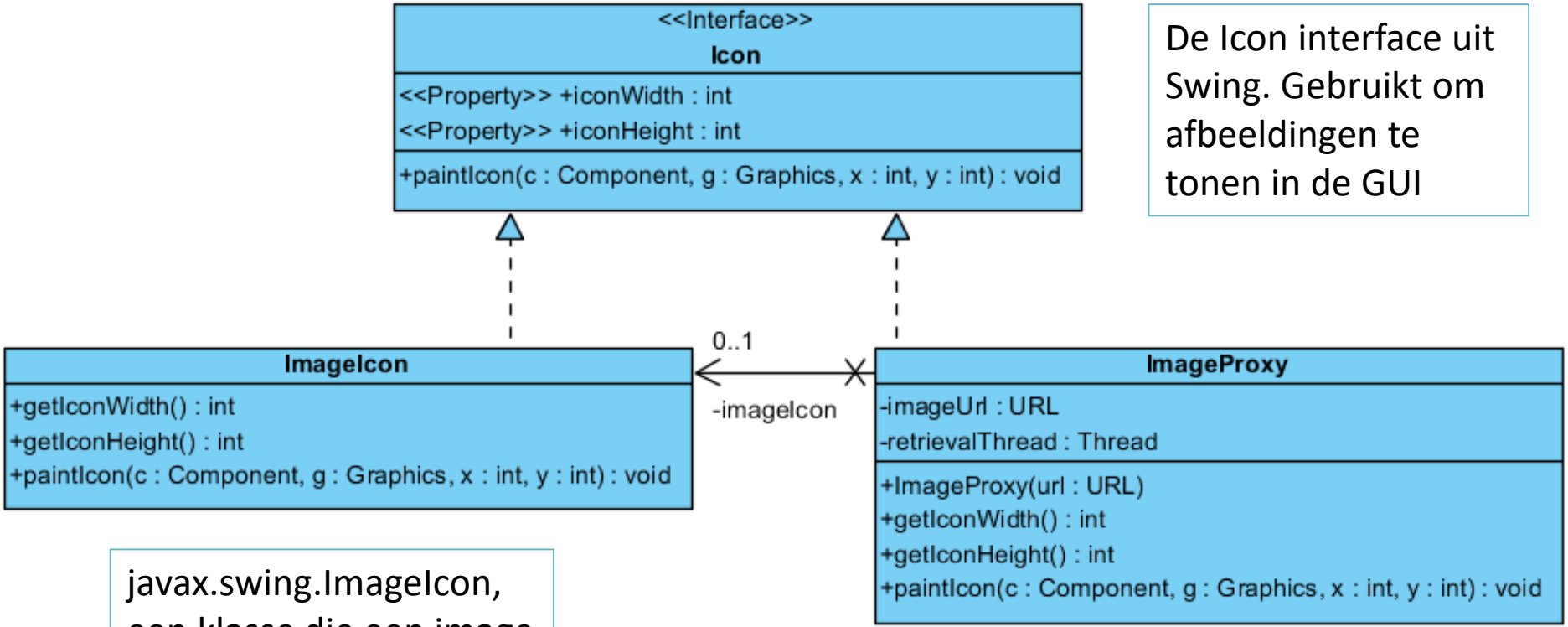
Kies de albumcover die je wil zien



Terwijl de cd cover wordt geladen (vb ophalen via een online service), toont de proxy een boodschap

Is de CD cover volledig geladen dan toont de proxy de afbeelding

Ontwerp van CD cover virtual proxy



De Icon interface uit Swing. Gebruikt om afbeeldingen te tonen in de GUI

javax.swing.ImageIcon, een klasse die een image toont

De proxy die eerst een boodschap toont en vervolgens wanneer de afbeelding geladen is, het tonen van de afbeelding delegeert aan ImageIcon

Ontwerp van CD cover virtual proxy

- Werking ImageProxy
  1. ImageProxy maakt eerst een ImageIcon en begint met het laden van een netwerk-URL
  2. Terwijl de bytes van de afbeelding worden opgehaald toont ImageProxy :” cd-cover wordt geladen, wachten a.u.b…”
  3. Wanneer de afbeelding volledig geladen is, delegeert ImageProxy alle methodeaanroepen naar ImageIcon, inclusief paintIcon, getWidth, getHeight
  4. Vraagt de gebruiker om een nieuwe afbeelding, dan maken we een nieuwe proxy en starten het proces opnieuw

De image proxy

```
import java.awt.Component;
import java.awt.Graphics;
import java.net.URL;
import javax.swing.Icon;
import javax.swing.ImageIcon;

public class ImageProxy implements Icon {

   private ImageIcon imageIcon;
   private URL imageUrl;
   private Thread retrievalThread;

   public ImageProxy(URL url) {
      this.imageUrl = url;
   }
```

ImageIcon is het echte pictogram dat eventueel de geladen afbeelding wil tonen

url = url van de afbeelding (die we tonen eens geladen)

De image proxy

```
@Override
public int getIconWidth() {
   if (imageIcon !=null)
      return imageIcon.getIconWidth();
   return 800;
}

@Override
public int getIconHeight() {
  if (imageIcon !=null)
      return imageIcon.getIconHeight();
   return 600;
}
```

Retourneer standaardhoogte en breedte totdat ImageIcon geladen is. Daarna dragen we dit over aan ImageIcon

De image proxy

Deze methode wordt aangeroepen als het tijd is om het pictogram op het scherm te tekenen

```
public void paintIcon(final Component c, Graphics g, int x, int y) {
  if (imageIcon != null) {
    imageIcon.paintIcon(c, g, x, y);
  } else {
    g.drawString("cd cover wordt geladen, wachten a.u.b. ...", x + 300, y + 190);
    if (retrievalThread==null) {
      retrievalThread = new Thread(new Runnable() {
        @Override
        public void run() {
          try {
            imageIcon = new ImageIcon(imageUrl, "cd-cover");
            c.repaint();
          } catch (Exception e) { e.printStackTrace(); }}
      });
      retrievalThread.start();
  } } }
```

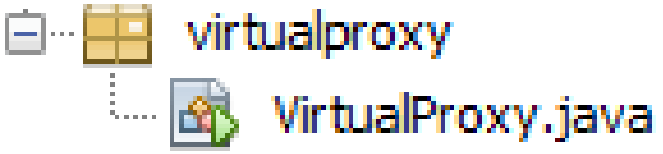
Hebben we al een pictogram, laat het zichzelf tekenen, anders toon de “laad”-boodschap

Als we nog niet proberen de afb. op te halen, doe het nu

We gebruiken een andere thread om het beeldje op te halen zodat de GUI niet blokkeert. In de thread instantiëren we het object Icon. Zijn constructor keert niet terug totdat de afbeelding geladen is.

Hebben we de afbeelding dan dragen we aan Swing op het scherm te hertekenen.

Eens uitproberen ....



- Een klein onderdeeltje, zie code voor de volledige methode

```
Icon icon = new ImageProxy(initialURL);
imageComponent = new ImageComponent(icon);
frame.getContentPane().add(imageComponent);
```

We maken een ImageProxy en geven deze een initiele url. Selecteer je een cd in het menu, dan krijg je een nieuwe url

Voeg proxy aan het frame toe, zodat dit getoond kan worden

Wikkel proxy in ImageComponent zodat deze aan het frame kan worden toegevoegd. De component zorgt voor de breedte en hoogte proxy

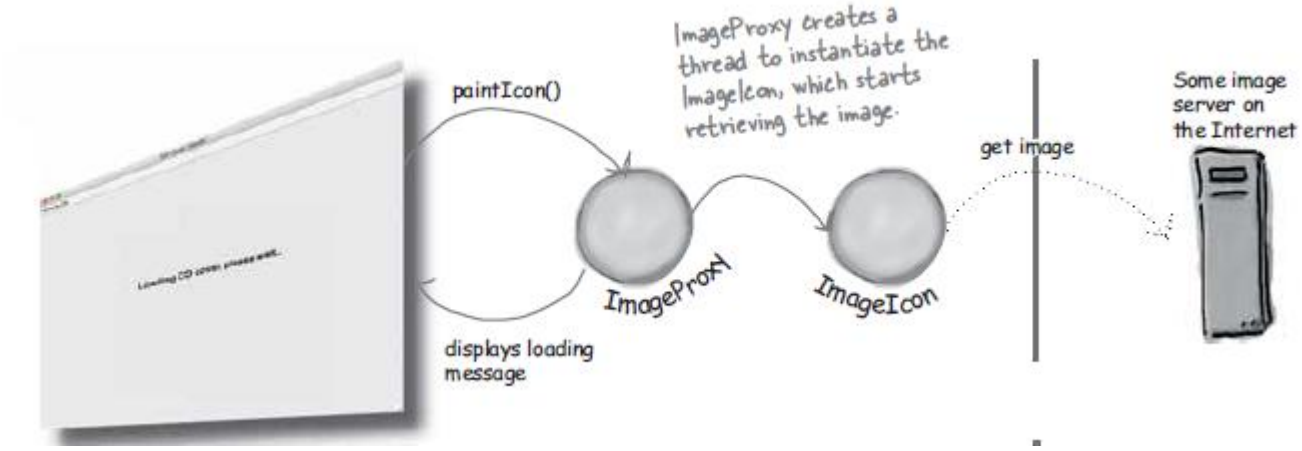
De image proxy

- Bekijk de code
  - De klasse kent meerdere toestanden. Via welk pattern kunnen we dit oplossen?

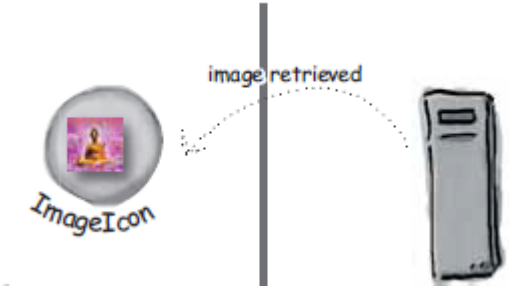
```
public int getIconWidth() {
   if (imageIcon != null) {  return imageIcon.getIconWidth();    }
   return 800;
}
@Override
public int getIconHeight() {
   if (imageIcon != null) {   return imageIcon.getIconHeight();      }
   return 600;
}

@Override
public void paintIcon(final Component c, Graphics g, int x, int y) {
   if (imageIcon != null) {
      imageIcon.paintIcon(c, g, x, y);
   } else {...
```

Wat hebben we gedaan

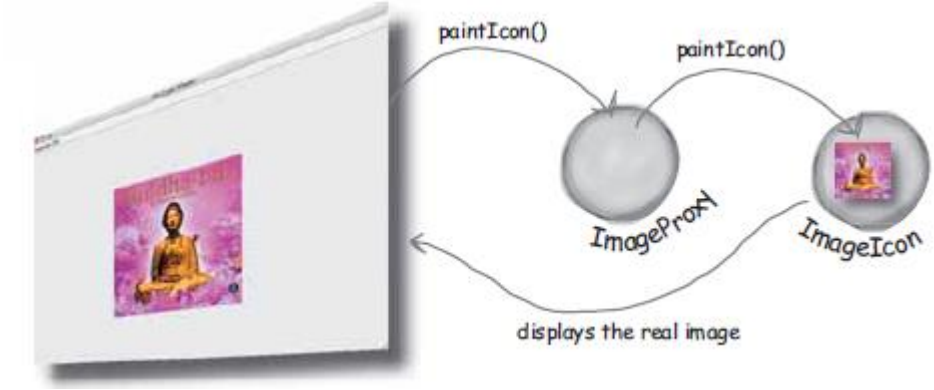
- 1.



- 2.

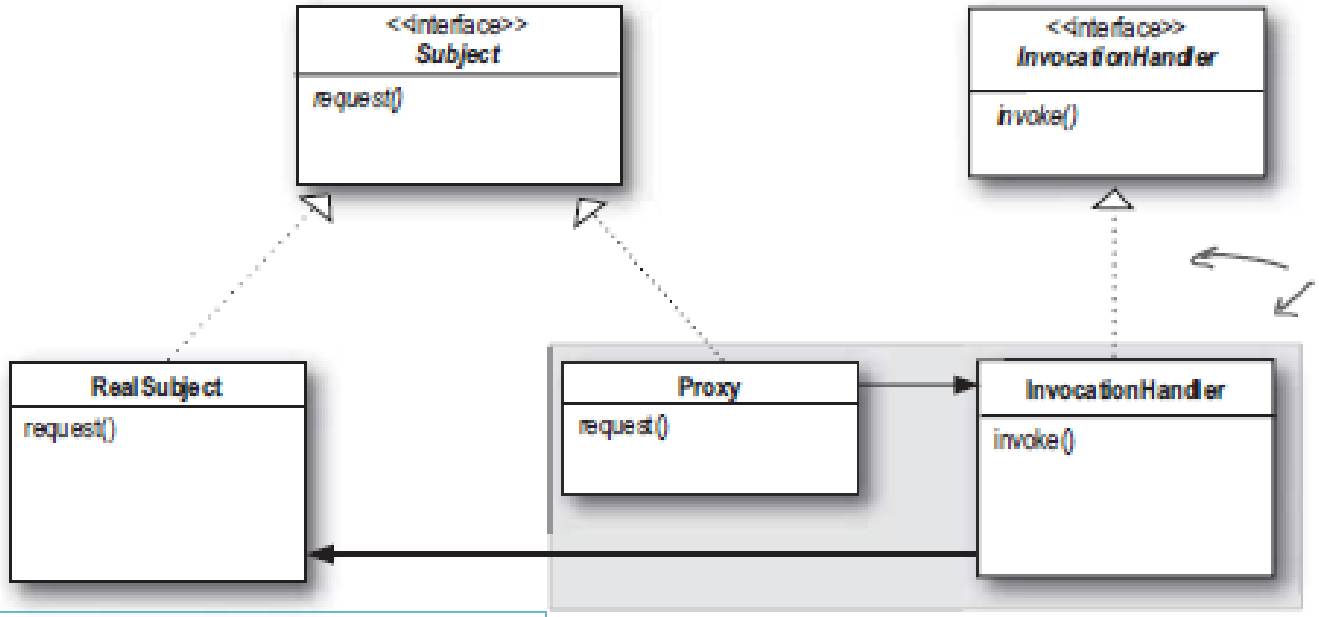


- 3.



Een protection proxy maken met de Proxy Api van Java

- De dynamic proxy uit java.lang.reflect



De PROXY bestaat nu uit 2 klassen

De PROXY wordt door Java gegenereerd en geïmplementeerd door de Subject interface

Je levert zelf de INVOCATIONHANDLER die alle methode aanroepen overgedragen krijgt die voor de Proxy worden gepleegd. Hij regelt de toegang tot de methoden van het RealSubject

Partnerbemiddeling in Objectville

- Een service voor partnerbemiddeling : draait om een PersonBean waarmee informatie over een persoon kan ingevoerd en opgevraagd worden

```
public interface PersonBean {

   public String getName();
   public String getGender();
   public String getInterests();
   public int getHotOrNotRating();

   public void setName(String string);
   public void setGender(String string);
   public void setInterests(String string);
   public void setHotOrNotRating(int i);
}
```

Partnerbemiddeling in Objectville

- De implementatie

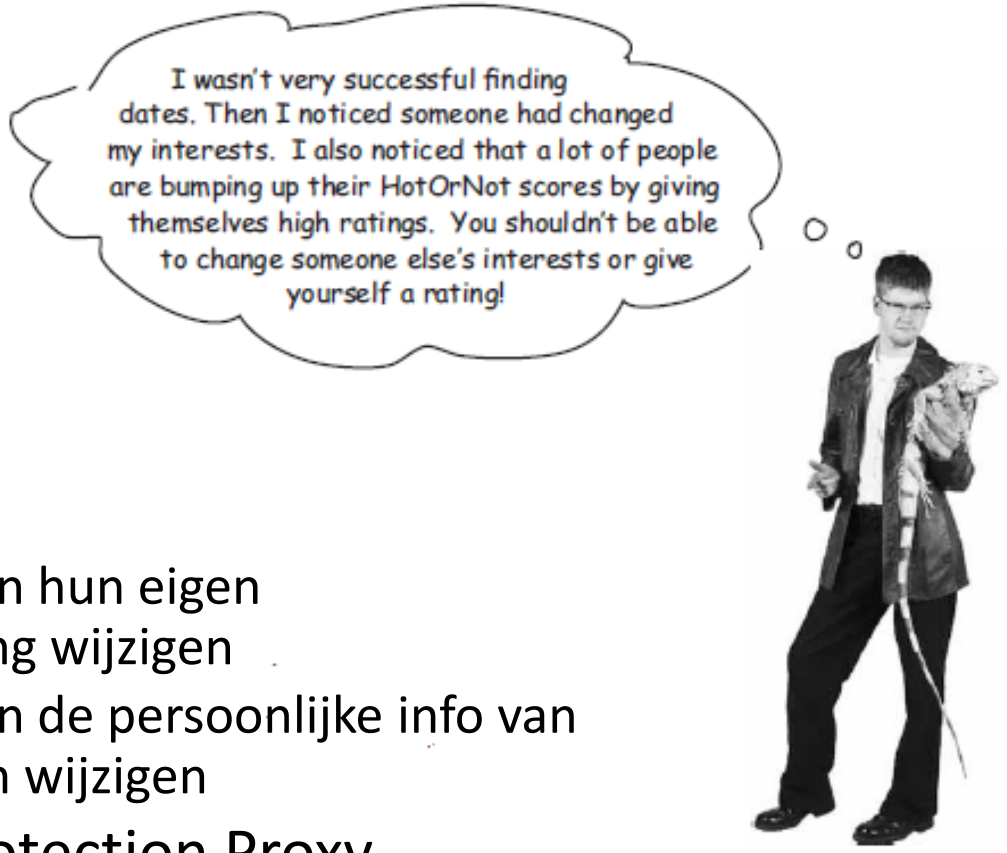
```
public class PersonBeanImpl implements PersonBean {
   private String name;
   private String gender;
   private String interests;
   private int rating;
   private int ratingCount;
   @Override
   public String getName() {   return name;   }
   @Override
   public String getGender() {  return gender;  }
   @Override
   public String getInterests() {  return interests; }
```

Partnerbemiddeling in Objectville

- De implementatie

```
public int getHotOrNotRating() {
   if (ratingCount == 0) {   return 0;  }
   return rating / ratingCount;
}
@Override
public void setName(String name) {   this.name = name;    }
@Override
public void setGender(String gender) {    this.gender = gender;    }
@Override
public void setInterests(String interests) {    this.interests = interests;   }
@Override
public void setHotOrNotRating(int rating) {
   this.rating += rating;
   ratingCount++;
}
```

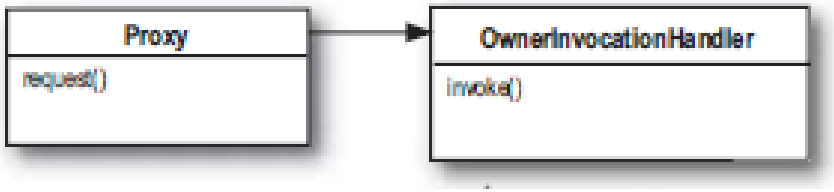
Partnerbemiddeling in Objectville



- Probleem
  - Klanten kunnen hun eigen HotOrNotRating wijzigen
  - Klanten kunnen de persoonlijke info van andere klanten wijzigen
- Oplossing : Protection Proxy
  - Regelt de toegang tot een object o.b.v. toegangsrechten

Een Dynamic Proxy voor PersonBean

- Maak 2 proxies
  - 1. Een proxy voor toegang tot je eigen PersonBean object



  - 2. Een proxy voor benaderen PersonBean object andere klanten

- De stappen
  - Stap 1 : maak 2 InvocationHandlers
  - Stap 2 : Schrijf de code die de dynamic proxies maakt
  - Stap 3 : Ontwikkel ieder PersonBean met de juiste proxy

Een Dynamic Proxy voor PersonBean

- Stap 1 : maak 2 InvocationHandlers
  - Wat is een InvocationHandler
    - Wanneer proxy een methode aanroep krijgt, stuurt hij die door naar de invocationhandler door aanroep methode invoke
    - 1. Stel de methode setHotOrNotRating() wordt voor de proxy aangeroepen
      - *proxy.setHotOrNotRating(9);*
    - 2. De proxy roept invoke aan bij invocationHandler
      - *invoke(Object proxy, Method method, Object[] args);*
    - De handler bepaalt wat hij met de aanvraag moet doen en kan deze doorsturen naar RealSubject.
      - *return method.invoke(person, args)*

Hier roepen we de originele methode aan die werd aangeroepen voor de proxy. Dit object werd doorgegeven met de invoke

Maar nu roepen we deze aan op het RealSubject

Een Dynamic Proxy voor PersonBean

- Stap 1 : maak 2 InvocationHandlers

RealSubject

```
import java.lang.reflect.*;
public class OwnerInvocationHandler implements InvocationHandler {
   private PersonBean person;
   public OwnerInvocationHandler(PersonBean person) {   this.person = person;    }
   public Object invoke(Object proxy, Method method, Object[] args)
         throws IllegalAccessException {
     try{
       if (method.getName().startsWith("get"))
           return method.invoke(person, args);
       if (method.getName().equals("setHotOrNotRating"))
           throw new IllegalAccessException();
       if (method.getName().startsWith("set"))
           return method.invoke(person, args);
     } catch (InvocationTargetException e) { e.printStackTrace();
     return null;
   }
```

Wordt aangeroepen bij een methode call voor de proxy :
-Bij een get gaan we verder en voeren deze uit
-Bij setHotOrNotRating, weigeren we deze
-Daar we eigenaar zijn, kunnen we iedere set methode aanroepen voor RealSubject

Een Dynamic Proxy voor PersonBean

- Stap 1 : maak 2 InvocationHandlers

```
import java.lang.reflect.*;
public class NonOwnerInvocationHandler {

}
```

Een Dynamic Proxy voor PersonBean

- Stap 2 : Proxyklasse maken en instantiëren proxyobject
  - De methode krijgt een object Person (het RealSubject) en retourneert een proxy hiervoor. Daar proxy dezelfde interface heeft als subject retourneren we een PersonBean

```
PersonBean getOwnerProxy(PersonBean person) {

   return (PersonBean) Proxy.newProxyInstance(
         person.getClass().getClassLoader(),
         person.getClass().getInterfaces(),
         new OwnerInvocationHandler(person));
}
```

Een static method van de Proxy klasse

De partnerbemiddelingsservice testen.

Stel dat Joe is ingelogd

```
public void drive() {
   PersonBean joe = getPersonFromDatabase("Joe Javabean");
   PersonBean ownerProxy = getOwnerProxy(joe);
   System.out.println("Name is " + ownerProxy.getName());
   ownerProxy.setInterests("bowling, Go");
   System.out.println("Interests set from owner proxy");
   try {
      ownerProxy.setHotOrNotRating(10);
   } catch (Exception e) {
      System.out.println("Can't set rating from owner proxy");
   }
   System.out.println("Rating is " + ownerProxy.getHotOrNotRating());
```

We halen joe op uit de db .. En maken er een eigenaarproxy voor

We roepen get… aan, dan set… en dan proberen we de rating te veranderen

De partnerbemiddelingsservice testen.

```
PersonBean kelly = getPersonFromDatabase("Kelly Klosure");
PersonBean nonOwnerProxy = getNonOwnerProxy(kelly);
System.out.println("Name is " + nonOwnerProxy.getName());
try {
    nonOwnerProxy.setInterests("bowling, Go");
} catch (Exception e) {
    System.out.println("Can't set interests from non owner proxy
}
nonOwnerProxy.setHotOrNotRating(3);
System.out.println("Rating set from non owner proxy");
System.out.println("Rating is " + nonOwnerProxy.getHotOrNotRating());
```

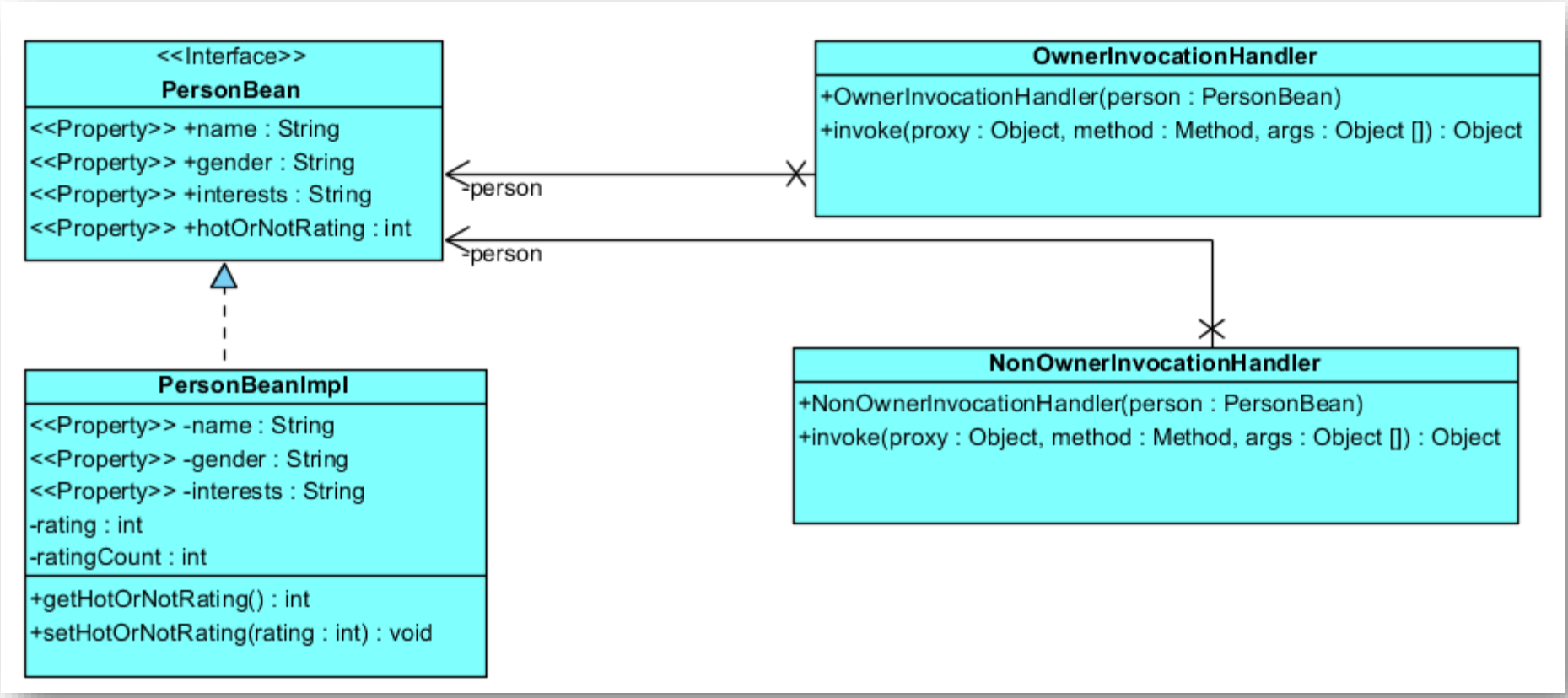
We halen Kelly op uit de db .. En maken er een NIET eigenaarproxy voor

We roepen get… aan, dan set… en dan proberen we de rating te veranderen

De partnerbemiddelingsservice testen.

```
run:
Name is Joe Javabean
Interests set from owner proxy
Can't set rating from owner proxy
Rating is 7
Name is Kelly Klosure
Can't set interests from non owner proxy
Rating set from non owner proxy
Rating is 4
```

De partnerbemiddelingsservice



De Proxy Zoo

Smart Reference Proxy provides additional actions whenever a subject is referenced, such as counting the number of references to an object.

Firewall Proxy controls access to a set of network resources, protecting the subject from "bad" clients.

Synchronization Proxy provides safe access to a subject from multiple threads.

Caching Proxy provides temporary storage for results of operations that are expensive. It can also allow multiple clients to share the results to reduce computation or network latency.

Complexity Hiding Proxy hides the complexity of and controls access to a complex set of classes. This is sometimes called the Facade Proxy for obvious reasons. The Complexity Hiding Proxy differs from the Facade Pattern in that the proxy controls access, while the Facade Pattern just provides an alternative interface.

Copy-On-Write Proxy controls the copying of an object by deferring the copying of an object until it is required by a client. This is a variant of the Virtual Proxy.

Proxy <-> Decorator

- En proxy is structureel gelijk aan een Decorator, maar hun doelstellingen verschillen
  - Decorator voegt gedrag toe aan een object, terwijl proxy de toegang regelt

Referentie

- Eric, F., & Elisabeth, F. (2004). *Head First Design Patterns* (p. 629). O'Reilly Media, Inc.